使用 Docker 部署 Spring Boot 项目

Docker

Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口，它是目前最流行的 Linux 容器解决方案。Docker 将应用程序与该程序的依赖打包在一个文件里面，运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样，有了 Docker，就不用担心环境问题了。

总体来说，Docker 的接口相当简单，用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

为什么要使用 Docker

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。

优势

- 更快速的交付和部署，开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。
- 更高效的虚拟化，Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。
- 更轻松的迁移和扩展，Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。
- 更简单的管理，使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。

Docker 相关概念

Docker 是 CS 架构，主要有两个概念，具体如下。

- Docker daemon：运行在宿主机上，Docker 守护进程，用户通过 Docker client（Docker 命令）与 Docker daemon 交互。
- Docker client：Docker 命令行工具，是用户使用 Docker 的主要方式，Docker client 与 Docker daemon 通信并将结果返回给用户，Docker client 也可以通过 socket 或者 RESTful

API 访问远程的 Docker daemon。

Docker 的组成有三个主要概念，具体如下。

- Docker image：镜像是只读的，镜像中包含有需要运行的文件。镜像用来创建 container，一个镜像可以运行多个 container；镜像可以通过 Dockerfile 创建，也可以从 Docker hub/registry 上下载。
- Docker container：容器是 Docker 的运行组件，启动一个镜像就是一个容器，容器是一个隔离环境，多个容器之间不会相互影响，保证容器中的程序运行在一个相对安全的环境中。
- Docker hub/registry：共享和管理 Docker 镜像，用户可以上传或者下载上面的镜像，官方地址可点击这里查看，也可以搭建自己私有的 Docker registry。

Spring Boot 项目

- 相关依赖：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
</parent>
```

添加 web 和测试依赖：

```
<dependencies>
     <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- 项目添加 Docker 支持

```

<properties>
    <docker.image.prefix>springboot</docker.image.prefix>
```

```
</properties>
```

- plugins 中添加 Docker 构建插件：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <!-- Docker maven plugin -->
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.0.0</version>
            <configuration>
                <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
                <dockerDirectory>src/main/docker</dockerDirectory>
                <resources>
                    <resource>
                        <targetPath>/</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.jar</include>
                    </resource>
                </resources>
            </configuration>
        </plugin>
        <!-- Docker maven plugin -->
    </plugins>
</build>
```

- ${docker.image.prefix}，自定义的镜像名称
- ${project.artifactId}，项目的 artifactId
- ${project.build.directory}，构建目录，缺省为 target
- $project.build.finalName$，产出物名称，缺省为{project.artifactId}-${project.version}

在目录 src/main/docker 下创建 Dockerfile 文件，Dockerfile 文件用来说明如何来构建镜像。

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD spring-boot-docker-1.0.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

- FROM，表示使用 JDK 8 环境为基础镜像，如果镜像不是本地的将会从 DockerHub 进行下

载。

- VOLUME，VOLUME 指向了一个 /tmp 的目录，由于 Spring Boot 使用内置的 Tomcat 容器，Tomcat 默认使用 /tmp 作为工作目录，这个命令的效果是：在宿主机的 /var/lib/docker 目录下创建一个临时文件并把它链接到容器中的 /tmp 目录。
- ADD，复制文件并且重命名。
- ENTRYPOINT，为了缩短 Tomcat 的启动时间，添加 java.security.egd 的系统属性指向 /dev/urandom 作为 ENTRYPOINT。

Dockerfile 介绍

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像的定制实际上就是定制每一层所添加的配置、文件。如果可以把每一层修改、安装、构建、操作的命令都写入一个脚本，用这个脚本来构建、定制镜像，那么之前提及的无法重复的问题、镜像构建透明性的问题、体积的问题就都会解决，这个脚本就是 Dockerfile。

Dockerfile 是一个文本文件，其内包含了一条条的指令（Instruction），每一条指令构建一层，因此每一条指令的内容，就是描述该层应当如何构建。有了 Dockerfile，当需要定制自己额外的需求时，只需在 Dockerfile 上添加或者修改指令，重新生成 image 即可，省去了敲命令的麻烦。

Dockerfile 文件格式如下：

```
##Dockerfile 文件格式

# This dockerfile uses the ubuntu image
# VERSION 2 - EDITION 1
# Author: docker_user
# Command format: Instruction [arguments / command] ..

# (1) 第一行必须指定基础镜像信息
FROM ubuntu

#  (2) 维护者信息
MAINTAINER docker_user docker_user@email.com

#  (3) 镜像操作指令
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/s
ources.list
```

```
RUN apt-get update && apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf

#  (4) 容器启动执行指令
CMD /usr/sbin/nginx
```

Dockerfile 分为四部分：基础镜像信息、维护者信息、镜像操作指令、容器启动执行指令。一开始必须要指明所基于的镜像名称，接下来一般会说明维护者信息；后面则是镜像操作指令，如 RUN 指令。每执行一条 RUN 指令，镜像添加新的一层，并提交；最后是 CMD 指令，来指明运行容器时的操作命令。

> 使用‘Docker 中国’加速器：

```
vi  /etc/docker/daemon.json

#添加后：
{
    "registry-mirrors": ["https://registry.docker-cn.com"],
    "live-restore": true
}
```